ODYNN Documentation

Marc Javin

Feb 05, 2019

Contents:

1 Objective **3**

2 Getting started **5**

3 Some examples **7**

4 Tutorials **9**

5 odynn package **11**
- 5.1 Structure . . . 11
- 5.2 Subpackages . . . 11
- 5.3 Submodules . . . 19
- 5.4 Module contents . . . 37

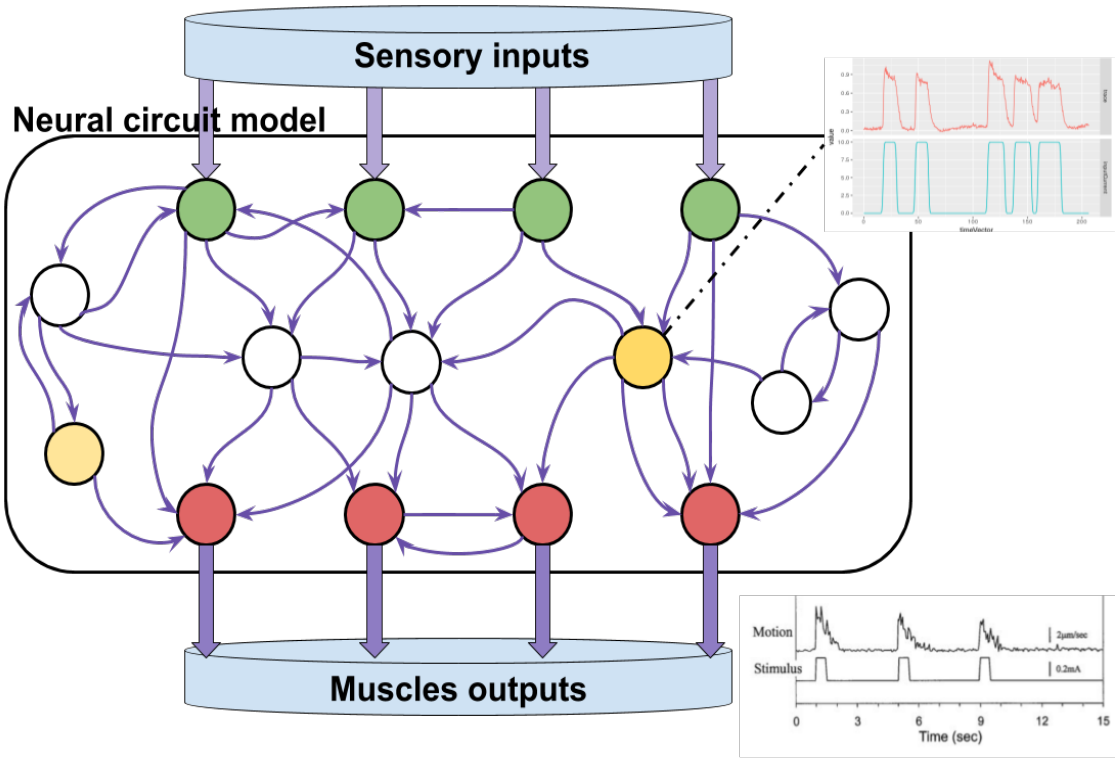
6 Indices and tables **39**

Python Module Index **41**

CHAPTER 1

Objective

ODYNN stands for Optimization for DYnamic Neural Network. ODYNN is designed for simulating and optimizing biological neural circuits to model multi-scale behaviors exhibited by the neural dynamics, and to test neuroscience related hypotheses. It is enhanced with features such as performing experiments with different biophysically realistic neuronal models as well as artificial recurrent neural networks (in particular long short term memory), incorporating calcium imaging data, to design arbitrarily structured neural circuits and optimizing them to govern specific behaviors.



CHAPTER 2

Getting started

Clone a local copy of this repository from Github using:

```
git clone https://github.com/MarcusJP/ODYNN
```

Go to the root directory and run:

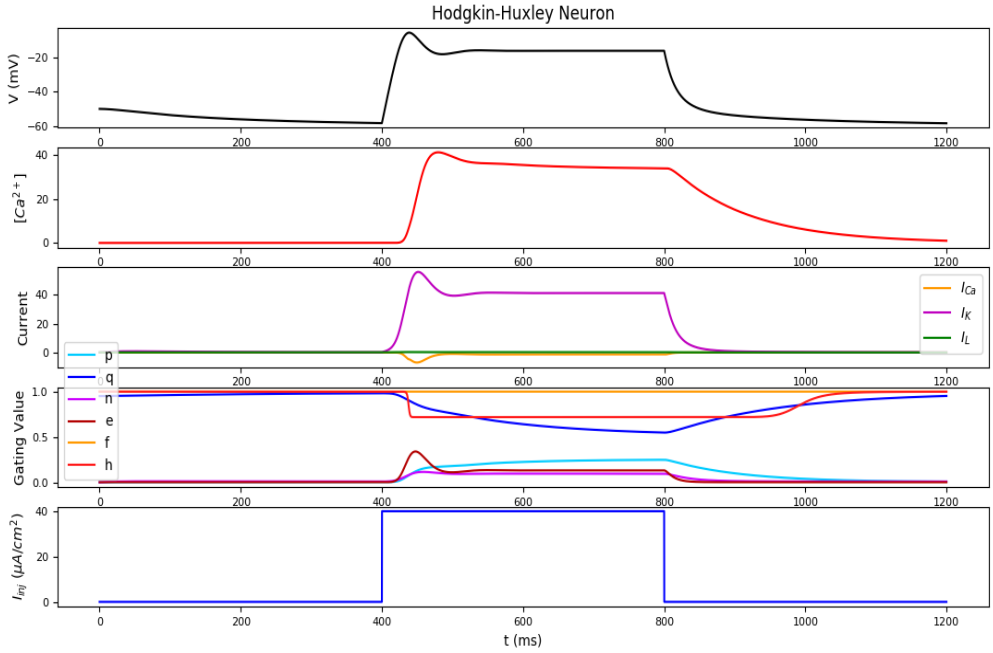
```
make init
make install
```

CHAPTER 3

Some examples

- Neuron simulation :

```
from odynn.nsimul import simul
import scipy as sp
t = sp.arange(0., 1200., 0.1)
i = 20. * ((t>400) & (t<800))
simul(t=t, i_inj=i, show=True)
```



- Neuron optimization :

```
import numpy as np
from odynn import utils, nsimul, neuron, noptim
#This file defines the model we will use
from odynn.models import cfg_model

dt = 1.
folder = 'Example'

# Function to call to set the target directories for plots and saved files
dir = utils.set_dir(folder)

#Definition of time and 2 input currents
t = np.arange(0., 1200., dt)
i_inj1 = 10. * ((t>200) & (t<600)) + 20. * ((t>800) & (t<1000))
i_inj2 = 5. * ((t>200) & (t<300)) + 30. * ((t>500) & (t<1000))
i_injs = np.stack([i_inj1, i_inj2], axis=-1)

#10 random initial parameters
params = [cfg_model.NEURON_MODEL.get_random() for _ in range(10)]
neuron = neuron.BioNeuronTf(params, dt=dt)

#This function will take the default parameters of the used model if none is given
train = nsimul.simul(t=t, i_inj=i_injs, show=True)

#Optimization
optimizer = noptim.NeuronOpt(neuron)
optimizer.optimize(dir=dir, train=train)
```

CHAPTER 4

Tutorials

Jupyter notebook tutorials are available. After cloning the repository, go to the *tutorial/* folder and run:

```
jupyter notebook
```

CHAPTER 5

odynn package

5.1 Structure

5.2 Subpackages

5.2.1 odynn.models package

Submodules

odynn.models.celeg module

class odynn.models.celeg.**CElegansNeuron**(*init_p=None*, *tensors=False*, *dt=0.1*)

Bases: *odynn.models.model.BioNeuron*

Full Hodgkin-Huxley Model implemented for C. elegans

Attributes

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

Methods

calculate(i_inj)	Simulate the neuron with input current *i_inj* and return the state vectors
get_random()	Returns a dictionnary of random parameters

Continued on next page

Table 1 – continued from previous page

parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(ts, i_inj_values, results[, ...])	plot all dynamics
plot_vars(var_dic[, suffix, show, save, func])	plot variation/comparison/boxplots of all variables organized by categories
plot_vars_gate(name, mdp, scale, tau, fig, ...)	plot the gates variables
step(X, i_inj)	Integrate and update state variable (voltage and possibly others) after one time step

boxplot_vars	
study_vars	

REST_CA = 0.0

static boxplot_vars(*var_dic*, *suffix=''*, *show=False*, *save=True*)

default_init_state = array([-6.0e+01, 0.0e+00, 9.5e-01, 0.0e+00, 0.0e+00, 1.0e+00, 1.0
initial state for neurons – voltage, rates and $[Ca^{2+}]$

default_params = {'C_m': 20.0, 'E_Ca': 20.0, 'E_K': -60.0, 'E_L': -60.0, 'decay_ca':
default parameters as a dictionnary

static get_random()
Returns a dictionnary of random parameters

plot_results(*ts*, *i_inj_values*, *results*, *ca_true=None*, *suffix=''*, *show=True*, *save=False*)
plot all dynamics

Parameters

- **ts** –
- **i_inj_values** –
- **results** –
- **ca_true** – (Default value = None)
- **suffix** – (Default value = "")
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)

Returns:

classmethod plot_vars(*var_dic*, *suffix='evolution'*, *show=False*, *save=True*, *func=<function plot>*)
plot variation/comparison/boxplots of all variables organized by categories

Parameters

- **var_dic** –
- **suffix** – (Default value = "")
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)

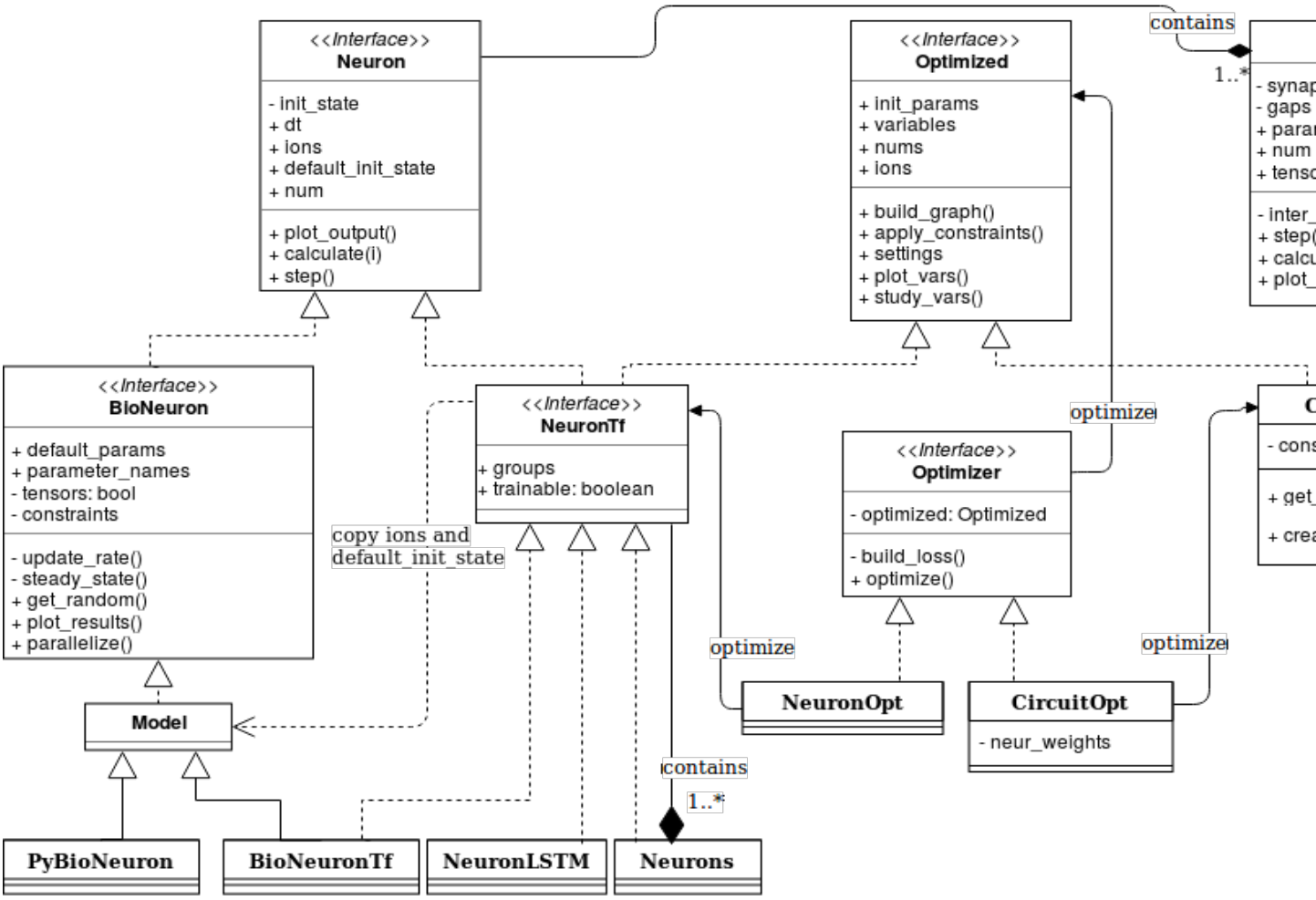


Fig. 1: ODYNN's class diagram

- **func** – (Default value = plot)

Returns:

static plot_vars_gate(*name*, *mdp*, *scale*, *tau*, *fig*, *pos*, *labs*, *func=<function plot>*)
plot the gates variables

Parameters

- **name** –
- **mdp** –
- **scale** –
- **tau** –
- **fig** –
- **pos** –
- **labs** –
- **func** – (Default value = plot)

Returns:

step(*X*, *i_inj*)
Integrate and update state variable (voltage and possibly others) after one time step

Parameters

- **X** (*ndarray*) – State variables
- **i** (*float*) – Input current

Returns updated state vector

Return type ndarray

classmethod study_vars(*p*, *suffix=''*, *target=None*, *show=False*, *save=True*)

odynn.models.celeg.**give_rand**()

odynn.models.cfg_model module

odynn.models.cfg_model.**NEURON_MODEL**
alias of *odynn.models.celeg.CElegansNeuron*

odynn.models.hhsimple module

class odynn.models.hhsimple.**HodgHuxSimple**(*init_p*, *tensors=False*, *dt=0.1*)
Bases: *odynn.models.model.BioNeuron*

Attributes

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

Methods

calculate(i_inj)	Simulate the neuron with input current *i_inj* and return the state vectors
get_random()	Return a dictionnary with the same keys as default_params and random values
parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(ts, i_inj_values, results[, ...])	Function for plotting detailed results of some experiment
step(X, i_inj)	Integrate and update state variable (voltage and possibly others) after one time step

default_init_state = array([-60., 0., 1.])

default_params = {'C_m': 1.0, 'E_K': 30.0, 'E_L': -60.0, 'a__mdp': -30.0, 'a__scale'

static get_random()
Return a dictionnary with the same keys as default_params and random values

plot_results(*ts*, *i_inj_values*, *results*, *ca_true=None*, *suffix=''*, *show=True*, *save=False*)
Function for plotting detailed results of some experiment

step(*X*, *i_inj*)
Integrate and update state variable (voltage and possibly others) after one time step

Parameters

- **X** (*ndarray*) – State variables
- **i** (*float*) – Input current

Returns updated state vector

Return type ndarray

odynn.models.leakint module

class odynn.models.leakint.**LeakyIntegrate**(*init_p*, *tensors=False*, *dt=0.1*)
Bases: *odynn.models.model.BioNeuron*

Attributes

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

Methods

calculate(i_inj)	Simulate the neuron with input current *i_inj* and return the state vectors

Continued on next page

Table 3 – continued from previous page

get_random()	Return a dictionnary with the same keys as default_params and random values
parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(ts, i_inj_values, X[, ca_true, ...])	Function for plotting detailed results of some experiment
step(X, i_inj)	Integrate and update state variable (voltage and possibly others) after one time step

default_init_state = array([-60.])

default_params = {'C_m': 1.0, 'E_L': -60.0, 'g_L': 0.1}

static get_random()
Return a dictionnary with the same keys as default_params and random values

plot_results(*ts*, *i_inj_values*, *X*, *ca_true=None*, *suffix=''*, *show=True*, *save=False*)
Function for plotting detailed results of some experiment

step(*X*, *i_inj*)
Integrate and update state variable (voltage and possibly others) after one time step

Parameters

- **X** (*ndarray*) – State variables
- **i** (*float*) – Input current

Returns updated state vector

Return type ndarray

odynn.models.model module

class odynn.models.model.**BioNeuron**(*init_p=None*, *tensors=False*, *dt=0.1*)
Bases: *odynn.models.model.Neuron*

Abstract class to implement for using a new biological model All methods and class variables have to be implemented in order to have the expected behavior

Attributes

default_init_state

default_params

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

Methods

calculate(i_inj)	Simulate the neuron with input current *i_inj* and return the state vectors
get_random()	Return a dictionnary with the same keys as default_params and random values
parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(*args, **kwargs)	Function for plotting detailed results of some experiment
step(X, i)	Integrate and update state variable (voltage and possibly others) after one time step

__init__(*init_p=None*, *tensors=False*, *dt=0.1*)
Reshape the initial state and parameters for parallelization in case init_p is a list

Parameters

- **init_p** (*dict or list of dict*) – initial parameters of the neuron(s). If init_p is a list, then this object will model n = len(init_p) neurons
- **tensors** (*bool*) – used in the step function in order to use tensorflow or numpy
- **dt** (*float*) – time step

calculate(*i_inj*)
Simulate the neuron with input current *i_inj* and return the state vectors

Parameters **i_inj** – input currents of shape [time, batch]

Returns series of state vectors of shape [time, state, batch]

Return type ndarray

default_params = None
dict, Default set of parameters for the model, of the form {<param_name> – value}

static get_random()
Return a dictionnary with the same keys as default_params and random values

parallelize(*n*)
Add a dimension of size n in the initial parameters and initial state

Parameters **n** (*int*) – size of the new dimension

parameter_names = None
names of parameters from the model

static plot_results(**args*, ***kwargs*)
Function for plotting detailed results of some experiment

class odynn.models.model.**Neuron**(*dt=0.1*)
Bases: abc.ABC

Attributes

default_init_state

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

Methods

calculate(i)	Iterate over i (current) and return the state variables obtained after each step
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
step(X, i)	Integrate and update state variable (voltage and possibly others) after one time step

V_pos = 0
int, Default position of the voltage in state vectors

calculate(*i*)
Iterate over i (current) and return the state variables obtained after each step

Parameters **i** (*ndarray*) – input current, dimension [time, (batch, (self.num))]

Returns state vectors concatenated [i.shape[0], len(self.init_state)(, i.shape[1], (i.shape[2]))]

Return type ndarray

default_init_state = None
array, Initial values for the vector of state variables

init_state
ndarray, Initial state vector

ions = {}

num
int, Number of neurons being modeled in this object

classmethod plot_output(*ts*, *i_inj*, *states*, *y_states=None*, *suffix=''*, *show=True*, *save=False*, *l=1*, *lt=1*, *targstyle='-'*)
Plot voltage and ion concentrations, potentially compared to a target model

Parameters

- **ts** (*ndarray of dimension [time]*) – time steps of the measurements
- **i_inj** (*ndarray of dimension [time]*) – input current
- **states** (*ndarray of dimension [time, state_var, nb_neuron]*) –
- **y_states** (*list of ndarray [time, nb_neuron], optional*) – list of values for the target model, each element is an ndarray containing the recordings of one state variable (Default value = None)
- **suffix** (*str*) – suffix for the name of the saved file (Default value = "")
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)
- **l** (*float*) – width of the main lines (Default value = 1)
- **lt** (*float*) – width of the target lines (Default value = 1)
- **targstyle** (*str*) – style of the target lines (Default value = '-')

step(*X*, *i*)
Integrate and update state variable (voltage and possibly others) after one time step

Parameters

- **X** (*ndarray*) – State variables
- **i** (*float*) – Input current

Returns updated state vector

Return type ndarray

Module contents

5.3 Submodules

5.3.1 odynn.circuit module

class odynn.circuit.**Circuit**(*neurons*, *synapses={}*, *gaps={}*, *tensors=False*, *labels=None*, *sensors=set()*, *commands=set()*)

Bases: object

Attributes

init_state (ndarray) Initial states of neurons

neurons Neurons contained in the circuit

num Number of circuits contained in the object, used to train in parallel

Methods

calculate(i_inj)	Simulate the circuit with a given input current.
plot([show, save])	Plot the circuit using networkx :param show: If True, show the figure :type show: bool :param save: If True, save the figure :type save: bool
plots_output_mult(ts, i_inj, states[, ...])	plot multiple voltages and Ca2+ concentration
step(hprev, curs)	run one time step

plot_output	

calculate(*i_inj*)

Simulate the circuit with a given input current.

Parameters **i_inj** (*ndarray*) – input current

Returns state vector and synaptical currents

Return type ndarray, ndarray

init_state

(ndarray) Initial states of neurons

neurons

Neurons contained in the circuit

num

Number of circuits contained in the object, used to train in parallel

parameter_names = ['E', 'G', 'mdp', 'scale', 'G_gap']
Circuit of neurons with synapses and gap junctions

plot(*show=True*, *save=False*)
Plot the circuit using networkx :param show: If True, show the figure :type show: bool :param save: If True, save the figure :type save: bool

plot_output(**args*, ***kwargs*)

plots_output_mult(*ts*, *i_inj*, *states*, *i_syn=None*, *suffix=''*, *show=True*, *save=False*, *l=1*, *trace=True*)
plot multiple voltages and Ca2+ concentration

Parameters

- **ts** (*ndarray*) – time sequence
- **i_inj** (*ndarray*) – input currents
- **states** (*ndarray*) – series of neural states
- **i_syn** (*ndarray, optional*) – synaptic currents (Default value = None)
- **suffix** (*str, optional*) – Suffix for the file name (Default value = "")
- **show** (*bool, optional*) – If True, show the figure (Default value = True)
- **save** (*bool, optional*) – If True, save the figure (Default value = False)

step(*hprev*, *curs*)
run one time step

For tensor :

Parameters

- **hprev** (*ndarray or tf.Tensor*) – previous state vector
- **curs** (*ndarray or tf.Tensor*) – input currents

Returns updated state vector

Return type ndarray or tf.Tensor

class odynn.circuit.**CircuitTf**(*neurons*, *synapses={}*, *gaps={}*, *labels=None*, *sensors=set()*, *commands=set()*)
Bases: *odynn.circuit.Circuit*, *odynn.optim.Optimized*

Circuit using tensorflow

Attributes

init_params (dict), initial model parameters

init_state (ndarray) Initial states of neurons

neurons Neurons contained in the circuit

num Number of circuits contained in the object, used to train in parallel

variables dict, current Tf variables

Methods

apply_constraints(session)	Apply the constraints and call the *apply_constraints* function of the neurons
build_graph([batch])	Build the tensorflow graph.
calculate(i)	Iterate over i (current) and return the state variables obtained after each step
plot([show, save])	Plot the circuit using networkx :param show: If True, show the figure :type show: bool :param save: If True, save the figure :type save: bool
plot_vars(var_dic[, suffix, show, save, func])	plot variation/comparison/boxplots of synaptic variables
plots_output_mult(ts, i_inj, states[, ...])	plot multiple voltages and Ca2+ concentration
reset()	prepare the variables as tensors, prepare the constraints, call reset for self._neurons
settings()	Returns(str): string describing the object
step(hprev, curs)	run one time step

apply_init	
create_random	
plot_output	
predump	
study_vars	

__init__(*neurons*, *synapses={}*, *gaps={}*, *labels=None*, *sensors=set()*, *commands=set()*)

Parameters

- **labels** –
- **synapses** (*dict*) – initial parameters for the synapses
- **neurons** (*NeuronModel*) – if not None, all other parameters except conns are ignores

apply_constraints(*session*)

Apply the constraints and call the *apply_constraints* function of the neurons

Parameters **session** – tensorflow session

apply_init(*session*)

build_graph(*batch=1*)

Build the tensorflow graph. Take care of the loop and the initial state.

calculate(*i*)

Iterate over i (current) and return the state variables obtained after each step

Parameters **i** (*ndarray*) – input current, [time, batch, neuron, model]

Returns state vectors concatenated [i.shape[0], len(self.init_state)(, i.shape[1]), self.num]

Return type ndarray

classmethod create_random(*n_neuron*, *syn_keys={}*, *gap_keys={}*, *n_rand=10*, *dt=0.1*, *labels=None*, *sensors=set()*, *commands=set()*, *fixed=()*, *groups=None*, *neurons=None*)

init_params

(dict), initial model parameters

plot_vars(*var_dic*, *suffix=''*, *show=True*, *save=False*, *func=<function plot>*)
plot variation/comparison/boxplots of synaptic variables

Parameters

- **var_dic** (*dict*) – synaptic parameters, each value of size [time, n_synapse, parallelization]
- **suffix** – (Default value = "")
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)
- **func** – (Default value = plot)

reset()
prepare the variables as tensors, prepare the constraints, call reset for self._neurons

settings()
Returns(str): string describing the object

study_vars(*p*, *loss=None*, *show=False*, *save=True*)

variables
dict, current Tf variables

odynn.circuit.**const_E**(*exc=True*)

odynn.circuit.**get_gap_rand**()
Give random parameters dictionnary for a gap junction

Returns random parameters for a gap junction

Return type dict

odynn.circuit.**get_syn_rand**(*exc=True*)
Give random parameters dictionnary for a chemical synapse

Parameters **exc** (*bool*) – If True, give an excitatory synapse (Default value = True)

Returns random parameters for a chemical synapse

Return type dict

odynn.circuit.**give_constraints**(*conns*)
Give constraints for synaptic parameters

Parameters **conns** (*dict*) – dictionnary of chemical synapse parameters

Returns constraints

Return type dict

odynn.circuit.**give_constraints_gap**()
Give constraints for gap junction parameters

Returns constraints

Return type dict

odynn.circuit.**give_constraints_syn**(*conns*)
Give constraints for chemical synapse parameters

Parameters **conns** (*dict*) – dictionnary of synapse parameters

Returns constraints

Return type dict

5.3.2 odynn.coptim module

class odynn.coptim.**CircuitOpt**(*circuit*)

Bases: *odynn.optim.Optimizer*

Class for optimization of a neuronal circuit

Methods

optimize(subdir[, train, test, w, w_n, ...])	Optimize the neuron parameters
settings(w, train)	Give the settings of the optimization

plot_out	

__init__(*circuit*)

Parameters **circuit** (CircuitTf) – Circuit to be optimized

optimize(*subdir, train=None, test=None, w=(1, 0), w_n=None, epochs=700, l_rate=(0.9, 9, 0.95), suffix='', n_out=[1], evol_var=True, plot=True*)

Optimize the neuron parameters

Parameters

- **dir** (*str*) – path to the directory for the saved files
- **train** (*list of ndarray*) – list containing [time, input, voltage, ion_concentration] that will be used fitted dimensions : - time : [time]
  - input, voltage and concentration : [time, batch, neuron]
- **test** (*list of ndarray*) – same as train for the dimensions These arrays will be used fo testing the model (Default value = None)
- **w** (*list*) – list of weights for the loss, the first value is for the voltage and the following ones for the ion concentrations defined in the model. (Default value = [1, 0]:
- **epochs** (*int*) – Number of training steps (Default value = 700)
- **l_rate** (*tuple*) – Parameters for an exponential decreasing learning rate : (start, number of constant steps, exponent) (Default value = [0.1, 9, 0.92]:
- **suffix** (*str*) – suffix for the saved files (Default value = '')
- **n_out** (*list of int*) – list of neurons corresponding to the data in train and test

Returns neuron attribute after optimization

Return type *NeuronTf*

plot_out(*X*, *results*, *res_targ*, *suffix*, *step*, *name*, *i*)

settings(*w*, *train*)

Give the settings of the optimization

Parameters **w** (*tuple*) – weights for the loss of voltage and ions concentrations

Returns settings

Return type str

odynn.coptim.**plot_heatmap**(*m*, *name*, *suffix*, *labels*, *n_out=None*)

5.3.3 odynn.csimul module

odynn.csimul.**simul**(*t, i_injs, pars=None, synapses={}, gaps={}, circuit=None, n_out=[0], suffix='', show=False, save=True, labels=None*)

Simulate a circuit with input current *i_injs* and return the outputs of neurons contained in *n_out*

Parameters

- **t** (*ndarray*) – time
- **i_injs** (*ndarray*) – input currents of shape [time, batch, neuron]
- **pars** (*dict or list*) – parameters for the neurons
- **synapses** (*dict or list*) – parameters of the chemical synapses
- **gaps** (*dict or list*) – parameters of the gap junctions
- **circuit(** – obj: Circuit): if not None, ignore the 3 previous arguments
- **n_out** (*list*) – neurons which output have to be saved
- **suffix** (*str*) – suffix for the plots
- **show** (*bool*) – If True, show the plot
- **save** (*bool*) – If True, save the plot
- **labels** – labels for the circuit's neurons

Returns measurements as a list [time, input currents, [voltage(, calcium)]]

Return type list

5.3.4 odynn.datas module

odynn.datas.**check_alpha**(*show=True*)

study the hill equation

odynn.datas.**full4**(*dt=0.1*, *nb_neuron_zero=None*, *max_t=1200.0*)

odynn.datas.**full4_test**(*dt=0.1*, *nb_neuron_zero=None*, *max_t=1200.0*)

odynn.datas.**get_real_data**(*delta=500*, *final_time=4000.0*, *dt=0.2*, *show=False*)

dump real data into our format

Parameters

- **delta** – (Default value = 500)
- **final_time** – (Default value = 4000.)
- **dt** (*float*) – time step (Default value = 0.2)

Returns:

odynn.datas.**get_real_data_norm**(*file='data/AVAL{}.csv'*)

odynn.datas.**give_periodic**(*t*, *max_i*, *size*, *freq*)

odynn.datas.**give_test**(*dt=0.1*, *max_t=1200.0*)
time and currents for optimization

Parameters **dt** (*float*) – time step (Default value = DT)

Returns:

odynn.datas.**give_train**(*dt=0.1*, *nb_neuron_zero=None*, *max_t=1200.0*)
time and currents for optimization

Parameters

- **dt** (*float*) – time step (Default value = DT)
- **nb_neuron_zero** – (Default value = None)
- **max_t** – (Default value = 1200.)

Returns:

odynn.datas.**give_train2**(*dt=0.1*)

odynn.datas.**rd**()
random() -> x in the interval [0, 1).

odynn.datas.**test**()

5.3.5 odynn.neuron module

class odynn.neuron.**BioNeuronTf**(*init_p=None*, *dt=0.1*, *fixed=()*, *constraints=None*, *groups=None*, *n_rand=None*)
Bases: *odynn.models.celeg.CElegansNeuron*, *odynn.neuron.NeuronTf*

Class representing a neuron, implemented using Tensorflow. This class allows simulation and optimization, alone and in a Circuit. It can contain several neurons at the same time. Which in turn can be optimized in parallel, or be used to represent the entire neurons in a Circuit.

Attributes

groups list indicating the group of each neuron

hidden_init_state For behavioral models eg LSTM

init_params initial model parameters

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

trainable True if the object can be optimized

variables Current variables of the models

Methods

apply_constraints(session)	Apply the constraints to the object variables
build_graph([batch])	Build a tensorflow graph for running the neuron(s) on a series of input :param batch: dimension of the batch :type batch: int

Continued on next page

Table 9 – continued from previous page

calculate(i)	Iterate over i (current) and return the state variables obtained after each step
get_random()	Returns a dictionnary of random parameters
init(batch)	Method to implement whe initialization is needed, will be called before reset
parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(ts, i_inj_values, results[, ...])	plot all dynamics
plot_vars(var_dic[, suffix, show, save, func])	plot variation/comparison/boxplots of all variables organized by categories
plot_vars_gate(name, mdp, scale, tau, fig, ...)	plot the gates variables
reset()	rebuild tf variable graph
settings()	Returns(str): string describing the object
step(X, i_inj)	Integrate and update state variable (voltage and possibly others) after one time step

apply_init	
boxplot_vars	
predump	
set_init_param	
study_vars	

__init__(*init_p=None*, *dt=0.1*, *fixed=()*, *constraints=None*, *groups=None*, *n_rand=None*)

Initializer :param init_p: initial parameters of the neuron(s). If init_p is a list, then this object

will model n = len(init_p) neurons

Parameters

- **dt** (*float*) – time step
- **fixed** (*set*) – parameters that are fixed and will stay constant in case of optimization. if fixed == 'all', all parameters will be constant
- **constraints** (*dict of ndarray*) – keys as parameters name, and values as [lower_bound, upper_bound]

apply_constraints(*session*)

Apply the constraints to the object variables

Parameters **session** – tensorflow session

build_graph(*batch=None*)

Build a tensorflow graph for running the neuron(s) on a series of input :param batch: dimension of the batch :type batch: int

Returns input placeholder and results of the run

Return type tf.placeholder, tf.Tensor

calculate(*i*)

Iterate over i (current) and return the state variables obtained after each step

Parameters **i** (*ndarray*) – input current

Returns state vectors concatenated [i.shape[0], len(self.init_state)(, i.shape[1]), self.num]

Return type ndarray

groups
list indicating the group of each neuron Neurons with the same group share the same parameters

init_params
initial model parameters

parallelize(*n*)
Add a dimension of size n in the initial parameters and initial state

Parameters **n** (*int*) – size of the new dimension

reset()
rebuild tf variable graph

set_init_param(*name*, *value=None*)

settings()
Returns(str): string describing the object

trainable
True if the object can be optimized

variables
Current variables of the models

class odynn.neuron.**NeuronLSTM**(*nb_layer=1*, *layer_size=50*, *extra_ca=0*, *dt=0.1*, *vars_init=None*)

Bases: *odynn.neuron.NeuronTf*

Behavior model of a neuron using an LSTM network

Attributes

groups list indicating the group of each neuron

hidden_init_state Give the initial state needed for the LSTM network

init_params Initial model parameters

init_state ndarray, Initial state vector

num Number of neurons contained in the object, always 1 here

trainable boolean stating if the neuron can be optimized

variables dict, current Tf variables

Methods

apply_constraints(session)	Apply necessary constraints to the optimized variables
apply_init(sess)	Initialize the variables if loaded object
build_graph([batch])	Build the tensorflow graph.
calculate(i)	Iterate over i (current) and return the state variables obtained after each step
init(batch)	Method to implement whe initialization is needed, will be called before reset

Continued on next page

Table 10 – continued from previous page

plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_vars(var_dic, suffix, show, save)	A function to plot the variables of the optimized object
settings()	Returns(str): string describing the object
step(X, hprev, i_inj)	Update function

predump	
reset	
study_vars	

apply_init (*sess*)
Initialize the variables if loaded object

Parameters **sess** – tf.Session

build_graph (*batch=1*)
Build the tensorflow graph. Take care of the loop and the initial state.

calculate (*i*)
Iterate over i (current) and return the state variables obtained after each step

Parameters **i** (*ndarray*) – input current

Returns state vectors concatenated [i.shape[0], len(self.init_state)(, i.shape[1]), self.num]

Return type ndarray

hidden_init_state
Give the initial state needed for the LSTM network

init (*batch*)
Method to implement whe initialization is needed, will be called before reset

Parameters **batch** (*int*) – number of batches

init_params
Initial model parameters

num
Number of neurons contained in the object, always 1 here

predump (*sess*)

reset ()

settings ()
Returns(str): string describing the object

step (*X*, *hprev*, *i_inj*)
Update function

Parameters

- **X** (*Tensor*) – not used here, classical state
- **hprev** (*tuple of LSTMStateTuple*) – previous LSTM state
- **i_inj** (*Tensor*) – array of input currents, dimension [batch]

Returns Tensor containing the voltages in the first position

Return type Tensor

class odynn.neuron.**NeuronTf**(*dt=0.1*)

Bases: *odynn.models.model.Neuron*, *odynn.optim.Optimized*

Abstract class whose implementation allow single optimization as well as in a Circuit

Attributes

groups list indicating the group of each neuron

hidden_init_state For behavioral models eg LSTM

init_params dict, initial parameters

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

trainable boolean stating if the neuron can be optimized

variables dict, current Tf variables

Methods

apply_constraints(session)	Apply necessary constraints to the optimized variables
build_graph([batch])	Build the tensorflow graph.
calculate(i)	Iterate over i (current) and return the state variables obtained after each step
init(batch)	Method to implement whe initialization is needed, will be called before reset
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_vars(var_dic, suffix, show, save)	A function to plot the variables of the optimized object
settings()	Give a string describing the settings Returns(str): description
step(X, i)	Integrate and update state variable (voltage and possibly others) after one time step

apply_init	
predump	
study_vars	

default_init_state = array([-6.0e+01, 0.0e+00, 9.5e-01, 0.0e+00, 0.0e+00, 1.0e+00, 1.0

groups

list indicating the group of each neuron Neurons with the same group share the same parameters

hidden_init_state

For behavioral models eg LSTM

init(*batch*)

Method to implement whe initialization is needed, will be called before reset

Parameters **batch** (*int*) – number of batches

nb = 0

trainable
boolean stating if the neuron can be optimized

class odynn.neuron.**Neurons**(*neurons*)
Bases: *odynn.neuron.NeuronTf*

This class allow to use neurons from different classes inheriting NeuronTf in a same Circuit

Attributes

groups list indicating the group of each neuron

hidden_init_state For behavioral models eg LSTM

init_params dict, initial parameters

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

trainable boolean stating if the neuron can be optimized

variables dict, current Tf variables

Methods

apply_constraints(session)	Apply the constraints to the object variables
build_graph()	Build the tensorflow graph.
calculate(i)	Iterate over i (current) and return the state variables obtained after each step
init(batch)	call *init* method for all contained neuron objects
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_vars(var_dic, suffix, show, save)	A function to plot the variables of the optimized object
settings()	Returns(str): string describing the object
step(X, hidden, i)	Share the state and the input current into its embedded neurons

apply_init	
predump	
reset	
study_vars	

__init__(*neurons*)

Parameters **neurons** (*list*) – list of NeuronTf objects

Raises AttributeError – If all neurons don't share the same dt

apply_constraints(*session*)
Apply the constraints to the object variables

Parameters **session** – tensorflow session

apply_init(*session*)

build_graph()
Build the tensorflow graph. Take care of the loop and the initial state.

calculate(*i*)
Iterate over i (current) and return the state variables obtained after each step

Parameters **i** (*ndarray*) – input current

Returns state vectors concatenated [i.shape[0], len(self.init_state)(, i.shape[1]), self.num]

Return type ndarray

hidden_init_state
For behavioral models eg LSTM

init(*batch*)
call *init* method for all contained neuron objects

predump(*sess*)

reset()

settings()
Returns(str): string describing the object

step(*X*, *hidden*, *i*)
Share the state and the input current into its embedded neurons

Parameters

- **X** (*tf.Tensor*) – precedent state vector
- **i** (*tf.Tensor*) – input current

Returns next state vector

Return type ndarray

class odynn.neuron.**PyBioNeuron**(*init_p=None*, *dt=0.1*)
Bases: *odynn.models.celeg.CElegansNeuron*

Class representing a neuron, implemented only in Python This class allows simulation but not optimization

Attributes

init_state ndarray, Initial state vector

num int, Number of neurons being modeled in this object

parameter_names

Methods

calculate(i_inj)	Simulate the neuron with input current *i_inj* and return the state vectors
get_random()	Returns a dictionnary of random parameters
parallelize(n)	Add a dimension of size n in the initial parameters and initial state
plot_output(ts, i_inj, states[, y_states, ...])	Plot voltage and ion concentrations, potentially compared to a target model
plot_results(ts, i_inj_values, results[, ...])	plot all dynamics

Continued on next page

Table 13 – continued from previous page

plot_vars(var_dic[, suffix, show, save, func])	plot variation/comparison/boxplots of all variables organized by categories
plot_vars_gate(name, mdp, scale, tau, fig, ...)	plot the gates variables
step(X, i_inj)	Integrate and update state variable (voltage and possibly others) after one time step

boxplot_vars	
study_vars	

5.3.6 odynn.noptim module

class odynn.noptim.**NeuronOpt**(*neuron*)

Bases: *odynn.optim.Optimizer*

Class for optimization of a neuron

Methods

optimize(dir, train[, test, w, epochs, ...])	Optimize the neuron parameters
settings(w, train)	Give the settings of the optimization

plot_out	

__init__(*neuron*)

Initializer, takes a NeuronTf object as argument

Parameters **neuron** (NeuronTf) – Neuron to be optimized

optimize(*dir*, *train*, *test=None*, *w=(1, 0)*, *epochs=700*, *l_rate=(0.1, 9, 0.92)*, *suffix=''*, *step=None*, *reload=False*, *reload_dir=None*, *evol_var=True*, *plot=True*)

Optimize the neuron parameters

Parameters

- **dir** (*str*) – path to the directory for the saved files
- **train** (*list of ndarray*) – list containing [time, input, voltage, ion_concentration] that will be used fitted dimensions : - time : [time]
  - input, voltage and concentration : [time, batch]
- **test** (*list of ndarray*) – same as train for the dimensions These arrays will be used fo testing the model (Default value = None)
- **w** (*list*) – list of weights for the loss, the first value is for the voltage and the following ones for the ion concentrations defined in the model. (Default value = [1, 0]:
- **epochs** (*int*) – Number of training steps (Default value = 700)
- **l_rate** (*tuple*) – Parameters for an exponential decreasing learning rate : (start, number of constant steps, exponent) (Default value = [0.1, 9, 0.92]:
- **suffix** (*str*) – suffix for the saved files (Default value = '')

- **step** – (Default value = None)
- **reload** (*bool*) – If True, will reload the graph saved in reload_dir (Default value = False)
- **reload_dir** (*str*) – The path to the directory of the experience to reload (Default value = None)

Returns neuron attribute after optimization

Return type NeuronTf

plot_out(*X*, *results*, *res_targ*, *suffix*, *step*, *name*, *i*)

5.3.7 odynn.nsimul module

odynn.nsimul.**comp_neuron_trace**(*neuron*, *trace*, *i_inj=array([0., 0., 0., ..., 0., 0., 0.])*, *scale=False*, *suffix=''*, *show=True*, *save=False*)

Compare a neuron with a given measured trace after scaling

Parameters

- **neuron** (*NeuronModel object*) – neuron to compare
- **trace** – recordings to plot
- **dt** (*float*) – time step
- **i_inj** (*ndarray*) – input currents
- **scale** – (Default value = False)
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)

odynn.nsimul.**comp_neurons**(*neurons*, *i_inj=array([0., 0., 0., ..., 0., 0., 0.])*, *suffix=''*, *show=True*, *save=False*)

Compare different neurons on the same experiment

Parameters

- **neurons** (*list of object NeuronModel*) – neurons to compare
- **dt** (*float*) – time step
- **i_inj** (*ndarray*) – input currents
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)

odynn.nsimul.**comp_pars**(*ps*, *t=None*, *dt=0.1*, *i_inj=array([0., 0., 0., ..., 0., 0., 0.])*, *suffix=''*, *show=True*, *save=False*)

Compare different parameter sets on the same experiment

Parameters

- **ps** (*list of dict*) – list of parameters to compare
- **dt** (*float*) – time step
- **i_inj** (*ndarray*) – input currents
- **show** (*bool*) – If True, show the figure (Default value = True)
- **save** (*bool*) – If True, save the figure (Default value = False)

odynn.nsimul.**comp_pars_targ**(*p*, *p_targ*, *t=None*, *dt=0.1*, *i_inj=array([0., 0., 0., ..., 0., 0., 0.])*, *suffix=''*, *save=False*, *show=True*)

Compare parameter sets with a target

Parameters

- **p** (*dict or list of dict*) – parameter(s) to compare with the target
- **p_targ** (*dict*) – target parameters
- **dt** (*float*) – time step
- **i_inj** (*ndarray*) – input currents
- **suffix** (*str*) – suffix for the saved figure (Default value = '')
- **save** (*bool*) – If True, save the figure (Default value = False)
- **show** (*bool*) – If True, show the figure (Default value = True)

odynn.nsimul.**simul**(*p=None*, *neuron=None*, *t=None*, *dt=0.1*, *i_inj=array([0., 0., 0., ..., 0., 0., 0.])*, *suffix=''*, *show=False*, *save=True*, *ca_true=None*)

Main demo for the Hodgkin Huxley neuron model

Parameters

- **p** (*dict*) – parameters of the neuron to simulate
- **neuron** (*NeuronModel object*) – neuron to simulate
- **dt** – time step

Returns records

Return type list

5.3.8 odynn.optim module

class odynn.optim.**Optimized**(*dt*)

Bases: abc.ABC

Abstract class for object to be optimized. It could represent on or a set of neurons, or a circuit.

Attributes

init_params dict, initial parameters

num int, number of models

variables dict, current Tf variables

Methods

apply_constraints(session)	Apply necessary constraints to the optimized variables
build_graph([batch])	Build the tensorflow graph.
plot_vars(var_dic, suffix, show, save)	A function to plot the variables of the optimized object
settings()	Give a string describing the settings Returns(str): description

apply_init	
predump	
study_vars	

apply_constraints(*session*)
Apply necessary constraints to the optimized variables

Parameters **session** (*tf.Session*) –

apply_init(*session*)

build_graph(*batch=1*)
Build the tensorflow graph. Take care of the loop and the initial state.

init_params
dict, initial parameters

ions = {}

num
int, number of models

static plot_vars(*var_dic*, *suffix*, *show*, *save*)
A function to plot the variables of the optimized object

Parameters

- **var_dic** –
- **suffix** –
- **show** –
- **save** –

predump(*sess*)

settings()
Give a string describing the settings Returns(str): description

study_vars(*p*, **args*, ***kwargs*)

variables
dict, current Tf variables

class odynn.optim.**Optimizer**(*optimized*, *frequency=30*)
Bases: abc.ABC

Methods

settings(w, train)	Give the settings of the optimization

optimize	
plot_out	

__init__(*optimized*, *frequency=30*)

Parameters

- **optimized** (Optimized) –
- **epochs** –
- **frequency** –

optimize(*dir*, *train_=None*, *test_=None*, *w=None*, *epochs=700*, *l_rate=(0.1*, *9*, *0.92)*, *suffix=”*, *step=”*, *reload=False*, *reload_dir=None*, *yshape=None*, *evol_var=True*, *plot=True*)

plot_out(**args*, ***kwargs*)

settings(*w*, *train*)
Give the settings of the optimization

Parameters **w** (*tuple*) – weights for the loss of voltage and ions concentrations

Returns settings

Return type str

odynn.optim.**get_best_result**(*dir*, *i=-1*, *loss=False*)

Parameters **dir** (*str*) – path to the directory i: (Default value = -1)

Returns:

odynn.optim.**get_data**(*dir*)

odynn.optim.**get_model**(*dir*)

odynn.optim.**get_vars**(*dir*, *i=-1*, *loss=False*)
get dic of vars from dumped file

Parameters **dir** (*str*) – path to the directory i: (Default value = -1)

Returns:

odynn.optim.**get_vars_all**(*dir*, *i=-1*, *losses=False*)
get dic of vars from dumped file

Parameters **dir** (*str*) – path to the directory i: (Default value = -1)

Returns:

odynn.optim.**plot_loss_rate**(*losses*, *rates*, *losses_test=None*, *parallel=1*, *suffix=”*, *show=False*, *save=True*)
plot loss (log10) and learning rate

Parameters

- **losses** –
- **rates** –
- **losses_test** – (Default value = None)
- **parallel** – (Default value = 1)
- **suffix** – (Default value = “”)
- **show** (*bool*) – If True, show the figure (Default value = False)
- **save** – (Default value = True)

Returns:

5.3.9 odynn.utils module

odynn.utils.**bar**(*ax*, *var*, *good_val=None*)

odynn.utils.**box**(*df*, *cols*, *labels*)

odynn.utils.**boxplot**(*ax*, *var*)

odynn.utils.**clamp**(*val*, *minimum=0*, *maximum=255*)

Clamp *val* between *minimum* and *maximum*

Parameters

- **val** (*float*) – value to clamp
- **minimum** (*int*) – minimum
- **maximum** (*int*) – maximum

Returns clamped value

Return type int

class odynn.utils.**classproperty**(*fget*)

Bases: object

odynn.utils.**colorscale**(*hexstr*, *scalefactor*)

Scales a hex string by scalefactor. Returns scaled hex string.

odynn.utils.**plot**(*ax*, *var*, *good_val=None*)

odynn.utils.**save_show**(*show*, *save*, *name=''*, *dpi=500*)

Show and/or save the current plot in *utils.current_dir*/*name* :param show: If True, show the plot :type show: bool :param save: If True, save the plot :type save: bool :param name: Name for the saved file :type name: str :param dpi: quality :type dpi: int

odynn.utils.**set_dir**(*subdir*)

Set directory for saving files to *utils.RES_DIR*'+'*subdir* and create subfolders

Parameters **subdir** (*str*) – name of the directory

Returns complete path to the new directory

Return type str

5.4 Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

circuit, 19
cls, 16
config, 14
coptim, 23
csimul, 24

d

datas, 24

h

hhmodel, 11

n

neuron, 25
neuropt, 32
neursimul, 33

o

odynn, 37
odynn.circuit, 19
odynn.coptim, 23
odynn.csimul, 24
odynn.datas, 24
odynn.models, 19
odynn.models.celeg, 11
odynn.models.cfg_model, 14
odynn.models.hhsimple, 14
odynn.models.leakint, 15
odynn.models.model, 16
odynn.neuron, 25
odynn.noptim, 32
odynn.nsimul, 33
odynn.optim, 34
odynn.utils, 37
optimize, 34

u

utils, 37

Index

Symbols

__init__() (odynn.circuit.CircuitTf method), 21
__init__() (odynn.coptim.CircuitOpt method), 23
__init__() (odynn.models.model.BioNeuron method), 17
__init__() (odynn.neuron.BioNeuronTf method), 26
__init__() (odynn.neuron.Neurons method), 30
__init__() (odynn.noptim.NeuronOpt method), 32
__init__() (odynn.optim.Optimizer method), 35

A

apply_constraints() (odynn.circuit.CircuitTf method), 21
apply_constraints() (odynn.neuron.BioNeuronTf method), 26
apply_constraints() (odynn.neuron.Neurons method), 30
apply_constraints() (odynn.optim.Optimized method), 35
apply_init() (odynn.circuit.CircuitTf method), 21
apply_init() (odynn.neuron.NeuronLSTM method), 28
apply_init() (odynn.neuron.Neurons method), 30
apply_init() (odynn.optim.Optimized method), 35

B

bar() (in module odynn.utils), 37
BioNeuron (class in odynn.models.model), 16
BioNeuronTf (class in odynn.neuron), 25
box() (in module odynn.utils), 37
boxplot() (in module odynn.utils), 37
boxplot_vars() (odynn.models.celeg.CElegansNeuron static method), 12
build_graph() (odynn.circuit.CircuitTf method), 21
build_graph() (odynn.neuron.BioNeuronTf method), 26
build_graph() (odynn.neuron.NeuronLSTM method), 28
build_graph() (odynn.neuron.Neurons method), 30
build_graph() (odynn.optim.Optimized method), 35

C

calculate() (odynn.circuit.Circuit method), 19
calculate() (odynn.circuit.CircuitTf method), 21
calculate() (odynn.models.model.BioNeuron method), 17
calculate() (odynn.models.model.Neuron method), 18
calculate() (odynn.neuron.BioNeuronTf method), 26
calculate() (odynn.neuron.NeuronLSTM method), 28
calculate() (odynn.neuron.Neurons method), 31
CElegansNeuron (class in odynn.models.celeg), 11
check_alpha() (in module odynn.datas), 24
Circuit (class in odynn.circuit), 19
circuit (module), 19
CircuitOpt (class in odynn.coptim), 23
CircuitTf (class in odynn.circuit), 20
clamp() (in module odynn.utils), 37
classproperty (class in odynn.utils), 37
cls (module), 16
colorscale() (in module odynn.utils), 37
comp_neuron_trace() (in module odynn.nsimul), 33
comp_neurons() (in module odynn.nsimul), 33
comp_pars() (in module odynn.nsimul), 33
comp_pars_targ() (in module odynn.nsimul), 33
config (module), 14
const_E() (in module odynn.circuit), 22
coptim (module), 23
create_random() (odynn.circuit.CircuitTf class method), 21
csimul (module), 24

D

datas (module), 24
default_init_state (odynn.models.celeg.CElegansNeuron attribute), 12
default_init_state (odynn.models.hhsimple.HodgHuxSimple attribute), 15
default_init_state (odynn.models.leakint.LeakyIntegrate attribute), 16
default_init_state (odynn.models.model.Neuron attribute), 18
default_init_state (odynn.neuron.NeuronTf attribute), 29
default_params (odynn.models.celeg.CElegansNeuron attribute), 12
default_params (odynn.models.hhsimple.HodgHuxSimple attribute), 15

default_params (odynn.models.leakint.LeakyIntegrate attribute), 16
default_params (odynn.models.model.BioNeuron attribute), 17

F

full4() (in module odynn.datas), 24
full4_test() (in module odynn.datas), 24

G

get_best_result() (in module odynn.optim), 36
get_data() (in module odynn.optim), 36
get_gap_rand() (in module odynn.circuit), 22
get_model() (in module odynn.optim), 36
get_random() (odynn.models.celeg.CElegansNeuron static method), 12
get_random() (odynn.models.hhsimple.HodgHuxSimple static method), 15
get_random() (odynn.models.leakint.LeakyIntegrate static method), 16
get_random() (odynn.models.model.BioNeuron static method), 17
get_real_data() (in module odynn.datas), 24
get_real_data_norm() (in module odynn.datas), 24
get_syn_rand() (in module odynn.circuit), 22
get_vars() (in module odynn.optim), 36
get_vars_all() (in module odynn.optim), 36
give_constraints() (in module odynn.circuit), 22
give_constraints_gap() (in module odynn.circuit), 22
give_constraints_syn() (in module odynn.circuit), 22
give_periodic() (in module odynn.datas), 24
give_rand() (in module odynn.models.celeg), 14
give_test() (in module odynn.datas), 24
give_train() (in module odynn.datas), 25
give_train2() (in module odynn.datas), 25
groups (odynn.neuron.BioNeuronTf attribute), 27
groups (odynn.neuron.NeuronTf attribute), 29

H

hhmodel (module), 11
hidden_init_state (odynn.neuron.NeuronLSTM attribute), 28
hidden_init_state (odynn.neuron.Neurons attribute), 31
hidden_init_state (odynn.neuron.NeuronTf attribute), 29
HodgHuxSimple (class in odynn.models.hhsimple), 14

I

init() (odynn.neuron.NeuronLSTM method), 28
init() (odynn.neuron.Neurons method), 31
init() (odynn.neuron.NeuronTf method), 29
init_params (odynn.circuit.CircuitTf attribute), 21
init_params (odynn.neuron.BioNeuronTf attribute), 27
init_params (odynn.neuron.NeuronLSTM attribute), 28
init_params (odynn.optim.Optimized attribute), 35
init_state (odynn.circuit.Circuit attribute), 19
init_state (odynn.models.model.Neuron attribute), 18
ions (odynn.models.model.Neuron attribute), 18
ions (odynn.optim.Optimized attribute), 35

L

LeakyIntegrate (class in odynn.models.leakint), 15

N

nb (odynn.neuron.NeuronTf attribute), 29
Neuron (class in odynn.models.model), 17
neuron (module), 25
NEURON_MODEL (in module odynn.models.cfg_model), 14
NeuronLSTM (class in odynn.neuron), 27
NeuronOpt (class in odynn.noptim), 32
Neurons (class in odynn.neuron), 30
neurons (odynn.circuit.Circuit attribute), 19
NeuronTf (class in odynn.neuron), 29
neuropt (module), 32
neursimul (module), 33
num (odynn.circuit.Circuit attribute), 19
num (odynn.models.model.Neuron attribute), 18
num (odynn.neuron.NeuronLSTM attribute), 28
num (odynn.optim.Optimized attribute), 35

O

odynn (module), 37
odynn.circuit (module), 19
odynn.coptim (module), 23
odynn.csimul (module), 24
odynn.datas (module), 24
odynn.models (module), 19
odynn.models.celeg (module), 11
odynn.models.cfg_model (module), 14
odynn.models.hhsimple (module), 14
odynn.models.leakint (module), 15
odynn.models.model (module), 16
odynn.neuron (module), 25
odynn.noptim (module), 32
odynn.nsimul (module), 33
odynn.optim (module), 34
odynn.utils (module), 37
optimize (module), 34
optimize() (odynn.coptim.CircuitOpt method), 23
optimize() (odynn.noptim.NeuronOpt method), 32
optimize() (odynn.optim.Optimizer method), 36
Optimized (class in odynn.optim), 34
Optimizer (class in odynn.optim), 35

P

parallelize() (odynn.models.model.BioNeuron method), 17

parallelize() (odynn.neuron.BioNeuronTf method), 27
parameter_names (odynn.circuit.Circuit attribute), 19
parameter_names (odynn.models.model.BioNeuron attribute), 17
plot() (in module odynn.utils), 37
plot() (odynn.circuit.Circuit method), 20
plot_heatmap() (in module odynn.coptim), 24
plot_loss_rate() (in module odynn.optim), 36
plot_out() (odynn.coptim.CircuitOpt method), 23
plot_out() (odynn.noptim.NeuronOpt method), 33
plot_out() (odynn.optim.Optimizer method), 36
plot_output() (odynn.circuit.Circuit method), 20
plot_output() (odynn.models.model.Neuron class method), 18
plot_results() (odynn.models.celeg.CElegansNeuron method), 12
plot_results() (odynn.models.hhsimple.HodgHuxSimple method), 15
plot_results() (odynn.models.leakint.LeakyIntegrate method), 16
plot_results() (odynn.models.model.BioNeuron static method), 17
plot_vars() (odynn.circuit.CircuitTf method), 21
plot_vars() (odynn.models.celeg.CElegansNeuron class method), 12
plot_vars() (odynn.optim.Optimized static method), 35
plot_vars_gate() (odynn.models.celeg.CElegansNeuron static method), 14
plots_output_mult() (odynn.circuit.Circuit method), 20
predump() (odynn.neuron.NeuronLSTM method), 28
predump() (odynn.neuron.Neurons method), 31
predump() (odynn.optim.Optimized method), 35
PyBioNeuron (class in odynn.neuron), 31

R

rd() (in module odynn.datas), 25
reset() (odynn.circuit.CircuitTf method), 22
reset() (odynn.neuron.BioNeuronTf method), 27
reset() (odynn.neuron.NeuronLSTM method), 28
reset() (odynn.neuron.Neurons method), 31
REST_CA (odynn.models.celeg.CElegansNeuron attribute), 12

S

save_show() (in module odynn.utils), 37
set_dir() (in module odynn.utils), 37
set_init_param() (odynn.neuron.BioNeuronTf method), 27
settings() (odynn.circuit.CircuitTf method), 22
settings() (odynn.coptim.CircuitOpt method), 23
settings() (odynn.neuron.BioNeuronTf method), 27
settings() (odynn.neuron.NeuronLSTM method), 28
settings() (odynn.neuron.Neurons method), 31
settings() (odynn.optim.Optimized method), 35
settings() (odynn.optim.Optimizer method), 36
simul() (in module odynn.csimul), 24
simul() (in module odynn.nsimul), 34
step() (odynn.circuit.Circuit method), 20
step() (odynn.models.celeg.CElegansNeuron method), 14
step() (odynn.models.hhsimple.HodgHuxSimple method), 15
step() (odynn.models.leakint.LeakyIntegrate method), 16
step() (odynn.models.model.Neuron method), 18
step() (odynn.neuron.NeuronLSTM method), 28
step() (odynn.neuron.Neurons method), 31
study_vars() (odynn.circuit.CircuitTf method), 22
study_vars() (odynn.models.celeg.CElegansNeuron class method), 14
study_vars() (odynn.optim.Optimized method), 35

T

test() (in module odynn.datas), 25
trainable (odynn.neuron.BioNeuronTf attribute), 27
trainable (odynn.neuron.NeuronTf attribute), 30

U

utils (module), 37

V

V_pos (odynn.models.model.Neuron attribute), 18
variables (odynn.circuit.CircuitTf attribute), 22
variables (odynn.neuron.BioNeuronTf attribute), 27
variables (odynn.optim.Optimized attribute), 35